#2 Priority Papers WDA 6/4/01

Docket No.: 57454-060 **PATENT**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of :

Isao MINEMATSU :

Serial No.: : Group Art Unit:

Filed: March 29, 2001 : Examiner:

For: MICROPROCESSOR EXECUTING DATA TRANSFER BETWEEN MEMORY AND REGISTER AND DATA TRANSFER BETWEEN REGISTERS IN RESPONSE TO SINGLE PUSH/POP INSTRUCTION

CLAIM OF PRIORITY AND
TRANSMITTAL OF CERTIFIED PRIORITY DOCUMENT

Commissioner for Patents
Washington, DC 20231

Sir:

In accordance with the provisions of 35 U.S.C. 119, Applicant hereby claims the priority of:

Japanese Patent Application No. 2000-194033,
filed June 28, 2000

cited in the Declaration of the present application. A certified copy is submitted herewith.

Respectfully submitted,

MCDERMOTT, WILL & EMERY

Stephen A. Becker
Registration No. 26,527

600 13th Street, N.W.
Washington, DC 20005-3096
(202) 756-8000 SAB:dtb
Date: March 29, 2001
Facsimile: (202) 756-8087

57454-060
Mizumatsu
March 29, 2001
McDermott, Will & Emery

日 本 国 特 許 庁
PATENT OFFICE
JAPANESE GOVERNMENT

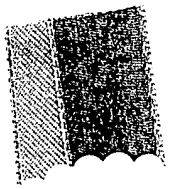
別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

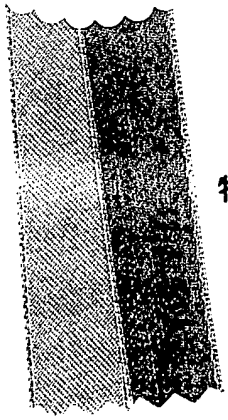
出 願 年 月 日
Date of Application: ２０００年　６月２８日

出 願 番 号
Application Number: 特願２０００－１９４０３３

出 願 人
Applicant (s): 三菱電機株式会社

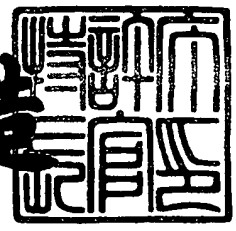


CERTIFIED COPY OF
PRIORITY DOCUMENT



２０００年　７月２１日

特 許 庁 長 官
Commissioner,
Patent Office 及 川 耕 造



出証番号　出証特２０００－３０５７７３３

【書類名】　特許願

【整理番号】　523187JP01

【提出日】　平成12年 6月28日

【あて先】　特許庁長官殿

【国際特許分類】　G06F 9/22

G06F 9/30

G06F 9/34

G06F 9/44

【発明者】

【住所又は居所】　東京都千代田区丸の内二丁目２番３号　三菱電機株式会社内

【氏名】　峯松　勲

【特許出願人】

【識別番号】　000006013

【氏名又は名称】　三菱電機株式会社

【代理人】

【識別番号】　100064746

【弁理士】

【氏名又は名称】　深見　久郎

【選任した代理人】

【識別番号】　100085132

【弁理士】

【氏名又は名称】　森田　俊雄

【選任した代理人】

【識別番号】　100091409

【弁理士】

【氏名又は名称】　伊藤　英彦

【選任した代理人】

【識別番号】　100096781

【弁理士】

【氏名又は名称】　堀井　豊

【選任した代理人】

【識別番号】　100096792

【弁理士】

【氏名又は名称】　森下　八郎

【手数料の表示】

【予納台帳番号】　008693

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　マイクロプロセッサ並びにアセンブラ、その方法およびそのプログラムを記録した記録媒体

【特許請求の範囲】

【請求項１】　命令コードのフェッチを制御するための制御手段と、

前記フェッチされた命令コードをデコードするためのデコード手段と、

前記デコード手段によるデコード結果に基づいて、メモリのアドレスを演算するためのアドレス演算手段と、

前記デコード手段によるデコード結果に基づいて、データを演算するためのデータ演算手段とを含み、

前記データ演算手段は、前記制御手段によってフェッチされた１つのオペレーションコードを有する１つの命令コードに対応して、レジスタ間のデータ転送と、レジスタとメモリとの間のデータ転送とを実行する、マイクロプロセッサ。

【請求項２】　前記データ演算手段は、前記制御手段によってフェッチされた１つの退避命令に対応して、第１のレジスタに格納されたデータをメモリに転送し、第２のレジスタに格納されたデータを前記第１のレジスタに転送する、請求項１記載のマイクロプロセッサ。

【請求項３】　前記データ演算手段は、前記第２のレジスタに格納されたデータを前記第１のレジスタに転送した後、スタックポインタの値をデクリメントする、請求項２記載のマイクロプロセッサ。

【請求項４】　前記データ演算手段は、前記制御手段によってフェッチされた１つの復帰命令に対応して、第１のレジスタに格納されたデータを第２のレジスタに転送し、メモリに格納されたデータを前記第１のレジスタに転送する、請求項１記載のマイクロプロセッサ。

【請求項５】　前記データ演算手段は、前記メモリに格納されたデータを前記第１のレジスタに転送した後、スタックポインタの値をインクリメントする、請求項４記載のマイクロプロセッサ。

【請求項６】　前記第１のレジスタは、前記データ演算手段に実装されるワークレジスタである、請求項２～５のいずれかに記載のマイクロプロセッサ。

【請求項７】　前記第２のレジスタは、前記アドレス演算手段または前記制御手段に実装される制御レジスタである、請求項２～６のいずれかに記載のマイクロプロセッサ。

【請求項８】　前記データ演算手段は、前記制御手段によってフェッチされた１つの退避命令に対して、第１のレジスタに格納されたデータをメモリに転送し、スタックポインタの値をそのまま保持するための手段を含む、請求項１～７のいずれかに記載のマイクロプロセッサ。

【請求項９】　ソースプログラムからコードを読み込むためのコード読込手段と、

複数のレジスタを特定するための情報を記憶するための記憶手段と、

前記コード読込手段によって読み込まれたコードが第１のマクロ命令である場合、前記コードに含まれる複数のレジスタを特定するための情報を前記記憶手段に記憶し、前記複数のレジスタを退避するコードを生成するための第１のコード生成手段と、

前記コード読込手段によって読み込まれたコードが第２のマクロ命令である場合、前記記憶手段に記憶された複数のレジスタを特定する情報を参照して、前記複数のレジスタを復帰させるコードを生成するための第２のコード生成手段を含むアセンブラ。

【請求項１０】　前記第１のコード生成手段は、前記コード読込手段によって読み込まれたコードが第１のマクロ命令である場合、前記コードに含まれる複数のレジスタのうち、レジスタとメモリとの間のデータ転送の媒介として使用されるレジスタ以外のレジスタを退避するコードを生成する、請求項９記載のアセンブラ。

【請求項１１】　ソースプログラムからコードを読み込むステップと、

前記コードが第１のマクロ命令である場合、前記コードに含まれる複数のレジスタを特定するための情報を記憶し、前記複数のレジスタを退避するコードを生成するステップと、

前記読み込まれたコードが第２のマクロ命令である場合、前記記憶された複数のレジスタを特定する情報を参照して、前記複数のレジスタを復帰させるコード

を生成するステップを含むアセンブリ方法。

【請求項１２】　前記複数のレジスタを退避するコードを生成するステップは、前記読み込まれたコードが第１のマクロ命令である場合、前記読み込まれたコードに含まれる複数のレジスタのうち、レジスタとメモリとの間のデータ転送の媒介として使用されるレジスタ以外のレジスタを退避するコードを生成するステップを含む、請求項１１記載のアセンブリ方法。

【請求項１３】　コンピュータにアセンブリ方法を実行させるプログラムが記録されたコンピュータで読取可能な記録媒体であって、

前記アセンブリ方法は、ソースプログラムからコードを読み込むステップと、

前記コードが第１のマクロ命令である場合、前記コードに含まれる複数のレジスタを特定するための情報を記憶し、前記複数のレジスタを退避するコードを生成するステップと、

前記読み込まれたコードが第２のマクロ命令である場合、前記記憶された複数のレジスタを特定する情報を参照して、前記複数のレジスタを復帰させるコードを生成するステップを含む、アセンブリプログラムを記録した記録媒体。

【請求項１４】　前記複数のレジスタを退避するコードを生成するステップは、前記読み込まれたコードが第１のマクロ命令である場合、前記読み込まれたコードに含まれる複数のレジスタのうち、レジスタとメモリとの間のデータ転送の媒介として使用されるレジスタ以外のレジスタを退避するコードを生成するステップを含む、請求項１３記載のアセンブリプログラムを記録した記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、マイクロプロセッサおよびプログラムをそのマイクロプロセッサが実行可能な機械語に変換するアセンブラに関し、特に、スタックに対してデータを効率的に退避／復帰するマイクロプロセッサ並びにアセンブラ、その方法およびそのプログラムを記録した記録媒体に関する。

【０００２】

【従来の技術】

近年、パーソナルコンピュータ等の情報処理装置を始めとして、様々な電子機器にマイクロプロセッサが使用されている。一般に、マイクロプロセッサがプログラムを実行する場合、レジスタに格納された現在の値を一時的に退避するために、メモリの一部をスタック領域に割り当てる。このスタックに対するデータの退避／復帰は、ＬＩＦＯ（Last In First Out）方式によって行なわれる。

【０００３】

このようなマイクロプロセッサにおいては、最後にスタックに保存されたデータの位置を管理するためのレジスタとして、スタックポインタが使用される。このスタックポインタの実装方法として、専用のレジスタとしてマイクロプロセッサに実装される場合と、汎用レジスタのうち１本がスタックポインタとして使用される場合とが挙げられる。命令長が１６ビット以上のマイクロプロセッサにおいては、レジスタの内容をスタックに退避する操作、またはスタックに格納されたデータをレジスタへ復帰させる操作をマイクロプロセッサに行なわせるために、ソフトウェアによって対象となるレジスタとその操作とを指定するのが一般的である。

【０００４】

【発明が解決しようとする課題】

近年主流となりつつあるＲＩＳＣ（Reduced Instruction Set Computer）方式のマイクロプロセッサにおいて、マイクロプロセッサを構成するレジスタは、メモリに対して直接データの読み出し／書き込みが行なえるレジスタ（以下、データ用レジスタと呼ぶ。）と、メモリに対して直接データの読み出し／書き込みが行なえないレジスタ（以下、総称して制御レジスタと呼ぶ。）とに大きく分類される。

【０００５】

制御レジスタの内容をスタックに退避したり、スタックに格納されたデータを制御レジスタに復帰させたりする場合、制御レジスタからワークレジスタへ一旦データを転送した後にスタックに書き込むか、スタックに格納されたデータをワークレジスタに一旦読み出した後に制御レジスタへ転送する必要がある。したがって、データ用レジスタの内容をスタックに退避したり、スタックに格納された

データをデータ用レジスタに復帰させる場合と比較して、プログラムのステップ数が余分に必要になるという問題点があった。

【０００６】

一方、全てのレジスタがメモリに対して直接データの読み出し／書き込みが行なえるタイプのマイクロプロセッサにおいては、レジスタの内容をスタックに退避したり、スタックに格納されたデータをレジスタに復帰させたりする際のプログラムのステップ数は、上述したＲＩＳＣ方式のマイクロプロセッサと比較して少なくなる。しかし、レジスタを選択する回路構成が複雑になるため、ＲＩＳＣ方式のマイクロプロセッサと比較して、動作周波数を高くすることが難しくなるという問題点があった。

【０００７】

本発明は、上記問題点を解決するためになされたものであり、第１の目的は、ＲＩＳＣ方式のマイクロプロセッサに採用される回路構成を採用しつつ、複数の制御レジスタの退避／復帰を短いステップ数のプログラムで実現できるマイクロプロセッサを提供することである。

【０００８】

第２の目的は、簡単なマクロ命令で複雑なスタック操作を行なえるアセンブラ、その方法およびそのプログラムを記録した記録媒体を提供することである。

【０００９】

第３の目的は、複数の制御レジスタの退避／復帰の整合性を自動的に管理することが可能なアセンブラ、その方法およびそのプログラムを記録した記録媒体を提供することである。

【００１０】

【課題を解決するための手段】

請求項１に記載のマイクロプロセッサは、命令コードのフェッチを制御するための制御手段と、フェッチされた命令コードをデコードするためのデコード手段と、デコード手段によるデコード結果に基づいて、メモリのアドレスを演算するためのアドレス演算手段と、デコード手段によるデコード結果に基づいて、データを演算するためのデータ演算手段とを含み、データ演算手段は、制御手段によ

ってフェッチされた１つのオペレーションコードを有する１つの命令コードに対応して、レジスタ間のデータ転送と、レジスタとメモリとの間のデータ転送とを実行する。

【００１１】

１つの命令コードでレジスタ間のデータ転送と、レジスタとメモリとの間のデータ転送とを実行できるため、所定の処理を行なうプログラムのステップ数を削減することが可能となる。

【００１２】

請求項２に記載のマイクロプロセッサは、請求項１記載のマイクロプロセッサであって、データ演算手段は、制御手段によってフェッチされた１つの退避命令に対応して、第１のレジスタに格納されたデータをメモリに転送し、第２のレジスタに格納されたデータを第１のレジスタに転送する。

【００１３】

したがって、直接メモリにデータを退避できない第２のレジスタに格納されたデータの退避を、１命令分のスループットで実行することが可能となる。

【００１４】

請求項３に記載のマイクロプロセッサは、請求項２記載のマイクロプロセッサであって、データ演算手段は、第２のレジスタに格納されたデータを第１のレジスタに転送した後、スタックポインタの値をデクリメントする。

【００１５】

第２のレジスタに格納されたデータを第１のレジスタに転送した後、スタックポインタの値がデクリメントされるため、スタック操作をさらに容易に行なうことが可能となる。

【００１６】

請求項４に記載のマイクロプロセッサは、請求項１記載のマイクロプロセッサであって、データ演算手段は、制御手段によってフェッチされた１つの復帰命令に対応して、第１のレジスタに格納されたデータを第２のレジスタに転送し、メモリに格納されたデータを前記第１のレジスタに転送する。

【００１７】

したがって、直接メモリからデータを復帰できない第２のレジスタへのデータの復帰を、１命令分のスループットで実行することが可能となる。

【００１８】

請求項５に記載のマイクロプロセッサは、請求項４記載のマイクロプロセッサであって、データ演算手段は、メモリに格納されたデータを第１のレジスタに転送した後、スタックポインタの値をインクリメントする。

【００１９】

メモリに格納されたデータを第１のレジスタに転送した後、スタックポインタの値がインクリメントされるため、スタック操作をさらに容易に行なうことが可能となる。

【００２０】

請求項６に記載のマイクロプロセッサは、請求項２～５のいずれかに記載のマイクロプロセッサであって、第１のレジスタは、データ演算手段に実装されるワークレジスタである。

【００２１】

したがって、第２のレジスタとメモリとの間のデータ転送を、ワークレジスタを媒介として行なうことが可能となる。

【００２２】

請求項７に記載のマイクロプロセッサは、請求項２～６のいずれかに記載のマイクロプロセッサであって、第２のレジスタは、アドレス演算手段または制御手段に実装される制御レジスタである。

【００２３】

したがって、第１のレジスタを媒介として制御レジスタとメモリとの間のデータ転送を行なうことが可能となる。また、第１のレジスタをワークレジスタとしてデータ演算手段に実装し、第２のレジスタを制御レジスタとしてアドレス演算手段に実装することによって、ＲＩＳＣ方式のマイクロプロセッサの回路構成を採用することが可能となる。

【００２４】

請求項８に記載のマイクロプロセッサは、請求項１～７のいずれかに記載のマ

イクロプロセッサであって、データ演算手段は、制御手段によってフェッチされた１つの退避命令に対して、第１のレジスタに格納されたデータをメモリに転送し、スタックポインタの値をそのまま保持するための手段を含む。

【００２５】

第１のレジスタに格納されたデータをメモリに転送し、スタックポインタの値をそのまま保持する退避命令を実行可能とすることにより、スタックポインタが最後に退避されたデータを指し示すようにすることが可能となる。

【００２６】

請求項９に記載のアセンブラは、ソースプログラムからコードを読み込むためのコード読込手段と、複数のレジスタを特定するための情報を記憶するための記憶手段と、コード読込手段によって読み込まれたコードが第１のマクロ命令である場合、コードに含まれる複数のレジスタを特定するための情報を記憶手段に記憶し、複数のレジスタを退避するコードを生成するための第１のコード生成手段と、コード読込手段によって読み込まれたコードが第２のマクロ命令である場合、記憶手段に記憶された複数のレジスタを特定する情報を参照して、複数のレジスタを復帰させるコードを生成するための第２のコード生成手段とを含む。

【００２７】

第１のコード生成手段は、第１のマクロ命令から複数のレジスタを退避するコードを生成するので、１つのマクロ命令で複雑なスタック操作を扱うことが可能となる。また、第２のコード生成手段は、記憶手段に記憶された複数のレジスタを特定する情報を参照して、複数のレジスタを復帰させるコードを生成するので、１つのマクロ命令で複雑なスタック操作を扱うことができ、複数のレジスタの退避／復帰の整合性を自動的に管理することが可能となる。

【００２８】

請求項１０に記載のアセンブラは、請求項９記載のアセンブラであって、第１のコード生成手段は、コード読込手段によって読み込まれたコードが第１のマクロ命令である場合、コードに含まれる複数のレジスタのうち、レジスタとメモリとの間のデータ転送の媒介として使用されるレジスタ以外のレジスタを退避するコードを生成する。

【００２９】

したがって、レジスタとメモリとの間のデータ転送の媒介として使用されるレジスタが自動的にスタックに退避される場合に、冗長なコードが生成されるのを防止することが可能となる。

【００３０】

請求項１１に記載のアセンブリ方法は、ソースプログラムからコードを読み込むステップと、コードが第１のマクロ命令である場合、コードに含まれる複数のレジスタを特定するための情報を記憶し、複数のレジスタを退避するコードを生成するステップと、読み込まれたコードが第２のマクロ命令である場合、記憶された複数のレジスタを特定する情報を参照して、複数のレジスタを復帰させるコードを生成するステップとを含む。

【００３１】

第１のマクロ命令から複数のレジスタを退避するコードを生成するので、１つのマクロ命令で複雑なスタック操作を扱うことが可能となる。また、読み込まれたコードが第２のマクロ命令である場合、記憶された複数のレジスタを特定する情報を参照して、複数のレジスタを復帰させるコードを生成するので、１つのマクロ命令で複雑なスタック操作を扱うことができ、複数のレジスタの退避／復帰の整合性を自動的に管理することが可能となる。

【００３２】

請求項１２に記載のアセンブリ方法は、請求項１１記載のアセンブリ方法であって、複数のレジスタを退避するコードを生成するステップは、読み込まれたコードが第１のマクロ命令である場合、読み込まれたコードに含まれる複数のレジスタのうち、レジスタとメモリとの間のデータ転送の媒介として使用されるレジスタ以外のレジスタを退避するコードを生成するステップを含む。

【００３３】

したがって、レジスタとメモリとの間のデータ転送の媒介として使用されるレジスタが自動的にスタックに退避される場合に、冗長なコードが生成されるのを防止することが可能となる。

【００３４】

請求項１３に記載の記録媒体は、コンピュータにアセンブリ方法を実行させるプログラムが記録されたコンピュータで読取可能な記録媒体であって、アセンブリ方法は、ソースプログラムからコードを読み込むステップと、コードが第１のマクロ命令である場合、コードに含まれる複数のレジスタを特定するための情報を記憶し、複数のレジスタを退避するコードを生成するステップと、読み込まれたコードが第２のマクロ命令である場合、記憶された複数のレジスタを特定する情報を参照して、複数のレジスタを復帰させるコードを生成するステップとを含む。

【００３５】

第１のマクロ命令から複数のレジスタを退避するコードを生成するので、１つのマクロ命令で複雑なスタック操作を扱うことが可能となる。また、読み込まれたコードが第２のマクロ命令である場合、記憶された複数のレジスタを特定する情報を参照して、複数のレジスタを復帰させるコードを生成するので、１つのマクロ命令で複雑なスタック操作を扱うことができ、複数のレジスタの退避／復帰の整合性を自動的に管理することが可能となる。

【００３６】

請求項１４に記載の記録媒体は、請求項１３記載のアセンブリプログラムが記録された記録媒体であって、複数のレジスタを退避するコードを生成するステップは、読み込まれたコードが第１のマクロ命令である場合、読み込まれたコードに含まれる複数のレジスタのうち、レジスタ間のデータ転送の媒介として使用されるレジスタ以外のレジスタを退避するコードを生成するステップを含む。

【００３７】

したがって、レジスタとメモリとの間のデータ転送の媒介として使用されるレジスタが自動的にスタックに退避される場合に、冗長なコードが生成されるのを防止することが可能となる。

【００３８】

【発明の実施の形態】

（実施の形態１）

図１は、本発明の実施の形態１におけるマイクロプロセッサを構成するレジス

タセットを説明するための図である。本実施の形態においては、マイクロプロセッサのデータ長が１６ビットの場合について説明するが、これに限られるものではない。

【００３９】

図１（ａ）に示すレジスタＲ０～Ｒ３は、４本の演算ソースレジスタであり、演算のソースが格納される。レジスタＴＲ０～ＴＲ３は、４本のワークレジスタであり、アドレスやデータ（演算ソースや演算結果）が一時的に保存されるレジスタである。Ａ０およびＡ１は、それぞれ４０ビットのアキュムレータである。アキュムレータＡ０およびＡ１はそれぞれ、演算ソースまたは演算結果の上位１６ビットが保持されるＡ０ＨおよびＡ１Ｈと、演算ソースまたは演算結果の下位１６ビットが保持されるＡ０ＬおよびＡ１Ｌと、上位ビットからあふれたビットが保持される８ビット長のガードビットＡ０ＧおよびＡ１Ｇとを含む。

【００４０】

図１（ｂ）に示すレジスタＡＲ０～ＡＲ３は、４本のアドレスレジスタであり、メモリアクセスの際のアドレスが格納される。レジスタＡＭＤ０～ＡＭＤ３は、４本のアドレッシングモードレジスタであり、それぞれアドレスレジスタＡＲ０～ＡＲ３を用いたアドレッシングモードが格納される。レジスタＡＲ＿ＳＥＬは、ＡＲ割当レジスタであり、アドレスレジスタの選択に使用される。

【００４１】

レジスタＭＯＤ＿ＳおよびＭＯＤ＿Ｅは、それぞれモジュロアドレッシングのための制御レジスタである。ＭＯＤ＿Ｓはモジュロスタートアドレスを保持し、ＭＯＤ＿Ｅはモジュロエンドアドレスを保持する。ＳＰはスタックポインタであり、スタックの先頭アドレスを保持する。レジスタＡＲ＿ＰＡＧＥは、ページ指定レジスタであり、メモリをページ単位で指定する場合のページ先頭アドレスを格納する。

【００４２】

ＰＣはプログラムカウンタであり、マイクロプロセッサが現在実行しているプログラムのアドレスを保持する。ＰＳＷはプロセッサ状態語であり、マイクロプロセッサを制御するためのフラグ等を格納する。ＢＰＣおよびＢＰＳＷは、それ

ぞれバックアップ用のプログラムカウンタおよびプロセッサ状態語であり、割り込み等の事象が発生した場合にＰＣおよびＰＳＷの値がそれぞれ自動的にコピーされる。

【００４３】

ＤＰＣおよびＤＰＳＷは、それぞれデバッガ用のプログラムカウンタおよびプロセッサ状態語であり、デバッグ用割り込み事象が発生した場合にＰＣおよびＰＳＷの値が自動的にコピーされる。レジスタＰＣＬＩＮＫはリンクレジスタであり、サブルーチンからの戻りアドレスを保持する。ＬＰ＿ＣＴおよびＲＥＰ＿ＣＴはそれぞれループカウンタおよびリピートカウンタであり、ブロックリピートの回数および単一命令リピートの回数をそれぞれ保持する。

【００４４】

レジスタＬＰ＿ＳおよびＬＰ＿Ｅは、それぞれブロックリピートの先頭アドレスおよび終了アドレスを指定するレジスタである。レジスタＰＣ＿ＢＲＫは、ハードウェアブレークポイントを指定する際に利用されるレジスタである。レジスタＩＮＴ＿Ｓは、割り込みステータスレジスタである。レジスタＣＲ００～ＣＲ６３は、周辺Ｉ／Ｏ（Input/Output）との間のデータの入出力に利用されるＩ／Ｏマップドレジスタである。なお、レジスタＩＮＴ＿ＳおよびＣＲ００～ＣＲ６３は、外部に接続された入出力デバイスを制御する際に使用されるレジスタであり、本実施の形態に直接関連するものではないため、詳細な説明は行なわない。なお、以下の説明においては、１つの命令コードに１つのオペレーションコードが含まれるものとする。

【００４５】

図２は、本実施の形態におけるマイクロプロセッサの概略構成を示すブロック図である。このマイクロプロセッサは、命令メモリ４３からフェッチされた命令をデコードする命令デコード部３９と、命令メモリ４３に格納された命令のフェッチを制御するＰＣＵ（Program Control Unit）部４０と、Ｘメモリ４４またはＹメモリ４５にアクセスする際のアドレスを演算するＡＡＵ（Address Arithmetic Unit）部４１と、データの演算を行なうＤＡＵ（Data Arithmetic Unit）部４２とを含む。命令メモリ４３は、命令のバイナリコードを格納する。また、Ｘ

メモリ４４およびＹメモリ４５は、被演算値、演算結果等のデータを格納する。

【００４６】

命令デコード部３９は、命令メモリ４３からフェッチされた命令コードをデコードし、命令コードに応じた制御信号Ｐ４６、Ａ４７およびＤ４８をそれぞれＰＣＵ部４０、ＡＡＵ部４１およびＤＡＵ部４２へ出力する。ＰＣＵ部４０は、命令デコード部３９から出力された制御信号Ｐ４６に基づいて、次にフェッチすべき命令が格納されるアドレスを、アドレスバス５６を介して命令メモリ４３へ出力する。ＡＡＵ部４１は、命令デコード部３９から出力された制御信号Ａ４７に基づいて、必要に応じて読み出すべきデータが格納されるアドレスを生成し、アドレスバス５７を介してＸメモリ４４およびＹメモリ４５へ出力する。

【００４７】

ＤＡＵ部４２は、１７ビット×１７ビットの乗算を行なう乗算器４９、４０ビットの２つのデータに演算を行なうＡＬＵ（Arithmetic and Logic Unit）５０および４０ビットの入力データに対して左右に１６ビットのシフト演算を行なうシフタ５１を含む。ＤＡＵ部４２は、命令デコード部３９から出力された制御信号Ｄ４８に基づいて、上述したレジスタに保持された値や、データバス５３を介してＸメモリ４４またはＹメモリ４５から読み出した値に対して、乗算、加減算またはシフト演算等を行なう。

【００４８】

図１に示すレジスタセットは、図２に示すＰＣＵ部４０、ＡＡＵ部４１およびＤＡＵ部４２のいずれかに実装されている。ＰＣＵ部４０は、１３個のレジスタＰＣ、ＰＳＷ、ＢＰＣ、ＢＰＳＷ、ＤＰＣ、ＤＰＳＷ、ＰＣＬＩＮＫ、ＬＰ＿ＣＴ、ＲＥＰ＿ＣＴ、ＬＰ＿Ｓ、ＬＰ＿Ｅ、ＰＣ＿ＢＲＫおよびＩＮＴ＿Ｓを有するレジスタ群６０を含む。

【００４９】

ＡＡＵ部４１は、１３個のレジスタＡＲ０～ＡＲ３、ＡＭＤ０～ＡＭＤ３、ＡＲ＿ＳＥＬ、ＭＯＤ＿Ｓ、ＭＯＤ＿Ｅ、ＳＰおよびＡＲ＿ＰＡＧＥを有するレジスタ群６１を含む。また、ＤＡＵ部４２は、４個のレジスタＴＲ０～ＴＲ３を有するレジスタ群６２と、４個のレジスタＲ０～ＴＲ３を有するレジスタ群６３と

、２個のレジスタＡ０およびＡ１を有するレジスタ群６４とを含む。

【００５０】

ＤＡＵ部４２は、レジスタ群６３の各レジスタからデータを受け、そのデータをＭＰＹ４９の一方の入力、ＡＬＵ５０の一方の入力またはＳＦＴ５１の一方の入力へ転送するためのデータバスＤ１と、レジスタ群６３の各レジスタからデータを受け、そのデータをＭＰＹ４９の他方の入力、ＡＬＵ５０の他方の入力またはＳＦＴ５１の他方の入力へ転送するためのデータバスＤ２と、ＭＰＸ４９、ＡＬＵ５０またはＳＦＴ５１から出力されたデータを受け、そのデータをレジスタ群６４の各レジスタへ転送するためのデータバスＤ３とを含む。ＤＡＵ部４２はさらに、データバスＤ６を含み、このデータバスＤ６を介してレジスタ群６４の各レジスタからＡＬＵ５０、ＳＦＴ５１のいずれかへのデータ転送を行なう。

【００５１】

また、ＤＡＵ部４２はさらにデータバスＤ４を含み、このデータバスＤ４を介してレジスタ群６３の各レジスタへ入力されるデータの転送、レジスタ群６４の各レジスタから出力されるデータの転送、およびレジスタ群６２の各レジスタに入出力されるデータの双方向の転送が行なわれる。さらには、このデータバスＤ４およびデータバス５３を介して、ＤＡＵ部４２とＸメモリ４４またはＹメモリ４５との間の双方向のデータ転送も行なわれる。

【００５２】

マイクロプロセッサはさらに、レジスタ群６０～６４間の双方向のデータ転送を行なうためのデータバスＤ５を含む。算術演算、論理演算、シフト等の演算命令を実行する場合には、レジスタ群６３または６４内の選択されたレジスタからデータバスＤ１および（または）Ｄ２またはＤ６へデータが供給され、その演算結果がデータバスＤ３を介してレジスタ群６４内の所定のレジスタへ格納される。

【００５３】

レジスタ間のデータ転送命令を実行する場合には、データバスＤ５を介してレジスタ間のデータ転送が行なわれる。特に、レジスタ群６４内のレジスタＡ０またはＡ１からレジスタ群６３内のレジスタへのデータ転送のときには、バスＤ４

が使用される。

【００５４】

ロード命令の場合には、Ｘメモリ４４またはＹメモリ４５のデータが、データバスＤ４を介してレジスタ群６２またはレジスタ群６３へ転送される。ストア命令の場合には、レジスタ群６２またはレジスタ群６４のデータが、データバスＤ４および５３を介してＸメモリ４４へ転送される。

【００５５】

次に、上述した本実施の形態におけるマイクロプロセッサが命令メモリ４３に格納されたプログラムを実行する手順について説明する。まず、ＰＣＵ部４０はフェッチすべき命令コードが格納されたアドレスを、アドレスバス５６を介して命令メモリ４３へ出力する。命令デコード部３９は、命令メモリ４３から出力された命令コードを、データバス５２を介して読み出して命令コードのデコードを行なう。命令デコード部３９は、命令コードのデコード結果に基づいて制御信号Ｐ４６、Ａ４７およびＤ４８を出力する。

【００５６】

ＰＣＵ部４０は、制御信号Ｐ４６に基づいて次にフェッチすべき命令が格納されるアドレスを生成し、アドレスバス５６を介して命令メモリ４３へ出力する。ＡＡＵ部４１は、制御信号Ａ４７がＸメモリ４４およびＹメモリ４５の一方または両方へのアクセスを示すものであれば、読み出しまたは書き込みのためのアドレスを生成し、アドレスバス５７を介してＸメモリ４４およびＹメモリ４５へ出力する。

【００５７】

ＤＡＵ部４２は、制御信号Ｄ４８に基づいて演算処理を行なう。たとえば、制御信号Ｄ４８がＸメモリ４４またはＹメモリ４５からデータを読み出して演算を行なうものであれば、ＤＡＵ部４２はＸメモリ４４またはＹメモリ４５から出力されたデータを読み込んで、そのデータに対して演算処理を行なう。また、制御信号Ｄ４８がレジスタの内容を演算し、演算結果をＸメモリ４４へ書き込むものであれば、ＤＡＵ部４２は演算結果をデータバス５３および５４を介してメモリ４４に出力する。

【００５８】

図４は、本実施の形態におけるマイクロプロセッサによって処理される演算命令の一例を示す図である。また、図５は、本実施の形態におけるマイクロプロセッサによって処理される転送命令、シーケンス制御命令および特殊命令の一例を示す図である。これらの命令の内容は、それぞれ命令の右側に説明されているので、詳細な説明は行なわない。

【００５９】

図４および図５に示す命令のうち、ＬＯＡＤ命令およびＳＴＯＲＥ命令は、アドレスレジスタＡＲ０～ＡＲ３を用いてアドレッシングを行ない、メモリとレジスタとの間のデータ転送を行なう命令である。このように、命令中にアドレスレジスタを指定するようにして、メモリのアクセスの自由度を高くしている。ただし、ＤＡＵ部４２内に実装されたレジスタの一部のみが指定可能である。そのため、これらの命令を用いてＤＡＵ部４２以外のブロックに実装されたレジスタの内容を退避するためには、レジスタ間転送を行なうｍｖ命令を用いて一旦ＤＡＵ部４２に実装されたレジスタに転送した後、レジスタとメモリとの間のデータ転送を行なう必要がある。図６は、ＬＯＡＤ命令およびＳＴＯＲＥ命令において指定可能なレジスタの一覧を示す図である。

【００６０】

図７は、ＳＴＯＲＥ命令を用いたレジスタ（ＴＲ０，ＡＲ０）の退避動作を行なうプログラムの一例を示す図である。図７の（１）および（２）は、アドレスレジスタＡＲ３およびそれに対応するアドレッシングモードレジスタＡＭＤ３を初期化している。すなわち、アドレスレジスタＡＲ３に＃ＳＴＡＣＫ＿ＢＯＴＴＯＭを代入し、アドレッシングモードレジスタＡＭＤ３に＃ＤＥＣ＿１を代入している。なお、＃ＳＴＡＣＫ＿ＢＯＴＴＯＭはレジスタ退避領域の最上位アドレスを表わす定数であり、＃ＤＥＣ＿１はＡＲ３の値が読み出される毎に１だけデクリメントされることを指定する定数を表わしている。

【００６１】

図７の（３）は、ワークレジスタＴＲ０の値が、Ｘメモリ４４のアドレスレジスタＡＲ３によって示されるアドレスに格納されることを示している。この命令

が実行されると、ＤＡＵ部４２に実装されているワークレジスタＴＲ０の値が、データバス５３および５４を介してＤＡＵ部４２からＸメモリ４４へ出力される。また、ＡＡＵ部４１に実装されているアドレスレジスタＡＲ３の値が、アドレスバス５７を介してＡＡＵ部４１からＸメモリ４４へ出力される。そして、ＡＡＵ部４１においてアドレスレジスタＡＲ３の値がデクリメントされる。

【００６２】

図７の（４）は、アドレスレジスタＡＲ０の値が、ワークレジスタＴＲ０に転送されることを示している。この命令が実行されると、ＡＡＵ部４１に実装されているアドレスレジスタＡＲ０の値が、ＤＡＵ部４２に実装されているワークレジスタＴＲ０に転送される。また、図７の（５）は（３）と同様に、ワークレジスタＴＲ０の値が、Ｘメモリ４４のアドレスレジスタＡＲ３によって示されるアドレスに格納されることを示している。

【００６３】

図８は、図７に示すプログラムが実行されたときのスタックの動作を示す図である。図７の（１）および（２）に示す命令コードが実行されると、アドレスレジスタＡＲ３に格納されるアドレスが図８（ａ）に示す位置を示すようになる。そして、図７の（３）に示す命令コードが実行されると、アドレスレジスタＡＲ３によって示されるアドレスにワークレジスタＴＲ０の値が格納され、アドレスレジスタＡＲ３の値がデクリメントされる。

【００６４】

図７の（４）に示す命令コードが実行されると、アドレスレジスタＡＲ０の値がワークレジスタＴＲ０に転送される（図８（ｃ）参照）。そして、図７の（５）に示す命令コードが実行されると、図８（ｄ）に示すように、アドレスレジスタＡＲ３によって示されるアドレスにワークレジスタＴＲ０（ＡＲ０）の値が格納され、アドレスレジスタＡＲ３の値がデクリメントされる。なお、ＬＯＡＤ命令を用いたレジスタへの復帰動作は、図７および図８に示す動作と逆の動作となるため、詳細な説明は行なわない。

【００６５】

図７および図８に示す転送方式において、制御レジスタの内容をスタックに退

避するためには、制御レジスタの内容をワークレジスタを介して転送する必要があるため、制御レジスタＮ個の内容を転送するためには（２×Ｎ）ステップ分の命令コードが必要となる。

【００６６】

本実施の形態におけるマイクロプロセッサにおいては、図１に示したレジスタセット以外に、次に説明するＰＯＰ命令、ＰＵＳＨ命令およびＰＵＴ命令を実装する。図９は、このＰＯＰ命令、ＰＵＳＨ命令およびＰＵＴ命令のニーモニックおよびその動作を説明するための図である。ＰＵＳＨ命令およびＰＯＰ命令は、任意のレジスタを指定することができる。また、ＰＵＴ命令は、レジスタを指定することができない。

【００６７】

図９（ａ）に示すように、ＰＯＰ命令においてレジスタ指定がない場合（１）には、Ｘメモリ４４のスタックポインタで指定されたアドレスに格納されるデータが、ＤＡＵ部４２に実装されたワークレジスタＴＲ０へ転送され、スタックポインタの値がインクリメントされる。また、ＰＯＰ命令においてレジスタ指定がある場合（２）には、ワークレジスタＴＲ０の値がＰＯＰ命令によって指定されるレジスタに転送され、Ｘメモリ４４のスタックポインタで指定されたアドレスに格納されるデータが、データバス５３および５４を介してＤＡＵ部４２に実装されたワークレジスタＴＲ０に転送された後、スタックポインタの値がインクリメントされる。

【００６８】

図９（ｂ）に示すように、ＰＵＳＨ命令においてレジスタ指定がない場合（１）には、スタックポインタの値がデクリメントされる。また、ＰＵＳＨ命令においてレジスタ指定がある場合（２）には、ＤＡＵ部４２に実装されたワークレジスタＴＲ０に格納されるデータが、Ｘメモリ４４のスタックポインタで指定されたアドレスに格納され、ＰＵＳＨ命令によって指定されたレジスタの値がワークレジスタＴＲ０に転送された後、スタックポインタの値がデクリメントされる。

【００６９】

図９（ｃ）に示すように、ＰＵＴ命令が実行されると、ＤＡＵ部４２に実装さ

れたワークレジスタＴＲ０に格納されるデータが、Ｘメモリ４４のスタックポインタで指定されたアドレスに転送される。なお、ＰＵＴ命令が実行されてもスタックポインタの値は更新されない。

【００７０】

なお、上述した説明において、ＰＯＰ命令のときにスタックポインタの値がインクリメントされ、ＰＵＳＨ命令のときにスタックポインタの値がデクリメントされる場合を説明したが、逆にＰＯＰ命令のときにスタックポインタの値がデクリメントされ、ＰＵＳＨ命令のときにスタックポインタの値がインクリメントされるようにしても良い。

【００７１】

上述したＰＯＰ命令、ＰＵＳＨ命令、ＰＵＴ命令の詳細動作を以下に説明する。図３は、本実施の形態におけるマイクロプロセッサのパイプライン処理を説明するための図である。これらの命令は、動作クロックの１サイクルずつ、それぞれ命令フェッチＩＦ、命令デコードＤおよび命令実行Ｅの３段のパイプラインで処理される。

【００７２】

オペランドありＰＯＰ命令は、時刻（Ａ）に示す動作クロックの立上りエッジでトリガされて、レジスタ群６２内のレジスタＴＲ０に記憶されたデータがデータバスＤ５に出力され、同時に、Ｘメモリ４４内のＳＰで示された領域に記憶されたデータがデータバス５３経由でＤ４に出力される。また、時刻（Ｂ）に示す動作クロックの立上りエッジでトリガされて、データバスＤ５に出力されたＴＲ０のデータが、レジスタ群６０、６１、６３および６４のうちＰＯＰ命令で指定されたレジスタに書込まれ、同時に、データバスＤ４に出力されたデータがＴＲ０に書込まれ、さらにＳＰの値がインクリメントされる。これらのデータ転送は、時刻（Ｃ）に示す動作クロックの立下がりエッジでされても良い。

【００７３】

オペランドなしＰＯＰ命令は、時刻（Ａ）に示す動作クロックの立上りエッジでトリガされて、Ｘメモリ４４内のＳＰで示された領域に記憶されたデータが、データバス５３経由でＤ４に出力される。また、時刻（Ｂ）に示す動作クロック

のエッジでトリガされて、データバスＤ４の値はＴＲ０に書込まれ、同時に、ＳＰの値はインクリメントされる。

【００７４】

オペランドありＰＵＳＨ命令は、時刻（Ａ）に示す動作クロックの立上りエッジでトリガされて、レジスタＴＲ０に記憶されたデータが、データバスＤ４経由でデータバス５３に出力されＸメモリ４４内のＳＰで示された領域に書込まれる。同時に、レジスタ群６０、６１、６３および６４のうちＰＵＳＨ命令で指定されたレジスタに記憶されたデータが、データバスＤ５に出力される。また、時刻（Ｂ）に示す動作クロックの立上りエッジでトリガされて、データバスＤ５に出力されたデータがＴＲ０に書込まれ、同時に、ＳＰの値はデクリメントされる。

【００７５】

オペランドなしＰＵＳＨ命令は、時刻（Ｂ）に示す動作クロックの立上りエッジでトリガされて、ＳＰの値がインクリメントされる。

【００７６】

ＰＵＴ命令は、時刻（Ａ）に示す動作クロックの立上りエッジでトリガされて、レジスタＴＲ０に記憶されたデータがデータバス５３およびＤ４を介してＸメモリ４４内のＳＰで示された領域へ書込まれる。

【００７７】

図１０は、上述したＰＯＰ命令、ＰＵＳＨ命令およびＰＵＴ命令を使用したプログラムの一例を示す図である。このプログラムは、演算ソースレジスタＲ０およびアドレスレジスタＡＲ０を退避した後、復帰するものである。図１０の（１）に示す“ｐｕｓｈ”は、スタックポインタの値をデクリメントして空き領域を指すことを示している。図１０の（２）に示す“ｐｕｓｈ　Ｒ０”は、ワークレジスタＴＲ０の値が、Ｘメモリ４４のスタックポインタで指定されたアドレスに格納され、演算ソースレジスタＲ０の値がワークレジスタＴＲ０に転送された後、スタックポインタの値がデクリメントされることを示している。

【００７８】

図１０の（３）に示す“ｐｕｓｈ　ＡＲ０”は、ワークレジスタＴＲ０の値が、Ｘメモリ４４のスタックポインタで指定されたアドレスに格納され、アドレス

レジスタＡＲ０の値がワークレジスタＴＲ０に転送された後、スタックポインタの値がデクリメントされることを示している。図１０の（４）に示す“ｐｕｔ”は、ワークレジスタＴＲ０の値が、Ｘメモリ４４のスタックポインタで指定されたアドレスに格納されることを示している。

【００７９】

図１０の（５）に示す“ｐｏｐ”は、Ｘメモリ４４のスタックポインタで示されたアドレスに格納されるデータが、ワークレジスタＴＲ０に転送された後、スタックポインタの値がインクリメントされることを示している。図１０の（６）に示す“ｐｏｐ　ＡＲ０”は、ワークレジスタＴＲ０の値がアドレスレジスタＡＲ０に転送され、Ｘメモリ４４のスタックポインタで示されたアドレスに格納されるデータが、ワークレジスタＴＲ０に転送された後、スタックポインタの値がインクリメントされることを示している。また、図１０の（７）に示す“ｐｏｐ　Ｒ０”は、ワークレジスタＴＲ０の値が演算ソースレジスタＲ０に転送され、Ｘメモリ４４のスタックポインタで示されたアドレスに格納されるデータが、ワークレジスタＴＲ０に転送された後、スタックポインタの値がインクリメントされることを示している。

【００８０】

図１１は、図１０に示すプログラムを実行したときのスタックの動作を説明すための図である。図１０に示すプログラムを実行する前は、図１１（ａ）に示す位置をスタックポインタが指し示している。図１０の（１）に示す命令を実行すると、スタックポインタの値がデクリメントされて、図１１（ｂ）に示すようにスタックポインタが空き領域を指し示すようになる。次に、図１０の（２）に示す命令が実行されると、ワークレジスタＴＲ０の値がＸメモリ４４のスタックポインタで示されたアドレスに格納される。そして、図１１（ｃ）に示すように演算ソースレジスタＲ０の値がワークレジスタＴＲ０に転送された後、スタックポインタの値がデクリメントされる。

【００８１】

次に、図１０の（３）に示す命令が実行されると、ワークレジスタＴＲ０の値（Ｒ０）がＸメモリ４４のスタックポインタで示されたアドレスに格納される。

そして、図１１（ｄ）に示すようにアドレスレジスタＡＲ０の値がワークレジスタＴＲ０に転送された後、スタックポインタの値がデクリメントされる。次に、図１０の（４）に示す命令が実行されると、ワークレジスタＴＲ０の値（ＡＲ０）がＸメモリ４４のスタックポインタで示されたアドレスに格納される。なお、このときスタックポインタの値は更新されない（図１１（ｅ）参照）。

【００８２】

次に、図１０の（５）に示す命令が実行されると、Ｘメモリ４４のスタックポインタで示されたアドレスに格納されるデータ（ＡＲ０）がワークレジスタＴＲ０に転送される。そして、図１１（ｆ）に示すようにスタックポインタの値がインクリメントされる。次に、図１０の（６）に示す命令が実行されると、ワークレジスタＴＲ０の値がアドレスレジスタＡＲ０に転送され、Ｘメモリ４４のスタックポインタで示されたアドレスに格納されるデータ（Ｒ０）がワークレジスタＴＲ０に転送される。そして、図１１（ｇ）に示すようにスタックポインタの値がインクリメントされる。

【００８３】

最後に、図１０の（７）に示す命令が実行されると、ワークレジスタＴＲ０の値が演算ソースレジスタＲ０に転送され、Ｘメモリ４４のスタックポインタで示されたアドレスに格納されるデータがワークレジスタＴＲ０に転送される。そして、図１１（ｈ）に示すようにスタックポインタの値がインクリメントされる。このように、ＰＵＳＨ命令、ＰＯＰ命令およびＰＵＴ命令を使用してスタック操作を行なうことにより、Ｎ個の制御レジスタの退避は（Ｎ＋２）ステップで実現でき、Ｎ個の制御レジスタの復帰は（Ｎ＋１）ステップで実現できる。

【００８４】

以上説明したように、本実施の形態におけるマイクロプロセッサによれば、１つの退避命令（ＰＵＳＨ命令）で制御レジスタからワークレジスタへのデータ転送と、ワークレジスタからＸメモリ４４へのデータ転送とが行なわれるようにし、１つの復帰命令（ＰＯＰ命令）でＸメモリ４４からワークレジスタへのデータ転送と、ワークレジスタから制御レジスタへのデータ転送とが行なわれるようにしたので、短いステップ数で複数の制御レジスタの退避および復帰が行なわれる

ようになった。また、退避命令および復帰命令によって上述した動作が行なわれるため、ＡＡＵ部４１に直接データバスを接続する必要がなくなり、ＲＩＳＣ方式のプロセッサの回路構成を採用することができ、マイクロプロセッサ内部の回路構成を簡略化することが可能となる。

【００８５】

（実施の形態２）

本発明の実施の形態２は、実施の形態１において説明したマイクロプロセッサが実行可能な機械語にプログラムを変換するアセンブラに関する。このアセンブラは、パーソナルコンピュータやワークステーション等のコンピュータにアセンブリプログラムを実行させることによって実現される。

【００８６】

図１２は、アセンブラを実現するコンピュータの構成例を示すブロック図である。このコンピュータは、コンピュータ本体１、グラフィックディスプレイ装置２、ＦＤ（Floppy Disk）４が装着されるＦＤドライブ３、キーボード５、マウス６、ＣＤ－ＲＯＭ（Compact Disc-Read Only Memory）８が装着されるＣＤ－ＲＯＭ装置７、およびネットワーク通信装置９を含む。

【００８７】

アセンブリプログラムは、ＦＤ４またはＣＤ－ＲＯＭ８等の記憶媒体によって供給される。アセンブリプログラムはコンピュータ本体１によって実行され、プログラマによって作成されたプログラムを実施の形態１において説明したマイクロプロセッサが実行可能な機械語に変換する。また、アセンブリプログラムは他のコンピュータより通信回線を経由し、コンピュータ本体１に供給されてもよい。

【００８８】

また、コンピュータ本体１は、ＣＰＵ１０、ＲＯＭ（Read Only Memory）１１、ＲＡＭ（Random Access Memory）１２およびハードディスク１３を含む。ＣＰＵ１０は、グラフィックディスプレイ装置２、磁気テープ装置３、キーボード５、マウス６、ＣＤ－ＲＯＭ装置７、ネットワーク通信装置９、ＲＯＭ１１、ＲＡＭ１２またはハードディスク１３との間でデータを入出力しながら処理を行なう

。ＦＤ４またはＣＤ－ＲＯＭ８に記録されたアセンブリプログラムは、ＣＰＵ１０によりＦＤドライブ３またはＣＤ－ＲＯＭ装置７を介して一旦ハードディスク１３に格納される。ＣＰＵ１０は、ハードディスク１３から適宜アセンブリプログラムをＲＡＭ１２にロードして実行することによって処理を行なう。

【００８９】

図１３は、本実施の形態におけるアセンブラによって処理されるマクロ命令を説明するための図である。図１３（ａ）の（１）に示すマクロ命令“ＭＰＵＳＨ　Ｒ０，ＡＲ０；”は、演算ソースレジスタＲ０およびアドレスレジスタＡＲ０を退避することを示している。なお、“ＭＰＵＳＨ”以降に退避すべきレジスタが記述されるものとし、レジスタの数は特に限定がないものとする。

【００９０】

図１３（ａ）の（２）に示すマクロ命令“ＭＰＯＰ”は、直前に記述されたマクロ命令“ＭＰＵＳＨ”に対応するマクロ命令であり、マクロ命令“ＭＰＵＳＨ”によって退避されたレジスタの内容を全て復帰させるものである。なお、マクロ命令の記述方法はこれらに限られるものではなく、マクロ命令が展開された後の命令コードが均等であれば同一と考えられるべきである。

【００９１】

図１３（ｂ）は、図１３（ａ）に示すマクロ命令“ＭＰＵＳＨ”を展開したときの命令コードを示している。また、図１３（ｃ）は、図１３（ａ）に示すマクロ命令“ＭＰＯＰ”を展開したときの命令コードを示している。図１３（ｂ）および図１３（ｃ）に示すプログラムの内容は、図１０に示すプログラムの内容と同じであるので、詳細な説明は繰り返さない。

【００９２】

図１４は、本実施の形態におけるアセンブラの機能的構成の概略を示すブロック図である。このアセンブラは、ソースファイルからコードを１行ずつ読み込み、読み込んだコードの解釈等を行なうコード解釈部２０と、ＭＰＵＳＨ命令を命令コードに展開するＭＰＵＳＨ命令展開部２１と、ＭＰＯＰ命令を命令コードに展開するＭＰＯＰ命令展開部２２と、ＭＰＵＳＨ命令およびＭＰＯＰ命令以外の命令のコード生成を行なうコード生成部２３とを含む。

【００９３】

図１５は、本実施の形態におけるアセンブラの処理手順を示している。まず、コード解釈部２０は、ソースファイルからコードを１行読み込み、そのソース行のコードにエラーがあるか否か、およびそのソース行が最終行であるか否かを判定する（Ｓ１）。そのソース行のコードにエラーがあるか、またはそのソース行が最終行であれば（Ｓ１，ＮＧ）、コード解釈部２０は処理を終了する。

【００９４】

また、そのソース行が最終行でなく、コードにエラーがなければ（Ｓ１，ＯＫ）、ソース解釈部２０はそのコードがＭＰＵＳＨ命令であるか否かを判定する（Ｓ２）。そのコードがＭＰＵＳＨ命令であれば（Ｓ２，Ｙｅｓ）、ＭＰＵＳＨ命令展開部２１はそのＭＰＵＳＨ命令を展開し（Ｓ３）、展開結果をＲＡＭ１２に格納する。その後、ステップＳ１へ戻って以降の処理を繰り返す。

【００９５】

また、コードがＭＰＵＳＨ命令でなければ（Ｓ２，Ｎｏ）、そのコードがＭＰＯＰ命令であるか否かを判定する（Ｓ４）。そのコードがＭＰＯＰ命令であれば（Ｓ４，Ｙｅｓ）、ＭＰＯＰ命令展開部２２はそのＭＰＯＰ命令を展開し（Ｓ５）、ステップＳ１へ戻って以降の処理を繰り返す。また、そのコードがＭＰＯＰ命令でなければ（Ｓ４，Ｎｏ）、コード生成部２３は通常のコード生成を行ない（Ｓ６）、そのコードをＲＡＭ１２に格納する。その後、ステップＳ１へ戻って以降の処理を繰り返す。

【００９６】

図１６は、図１５のステップＳ３（ＭＰＵＳＨ命令の展開）の処理をさらに詳細に説明するためのフローチャートである。まず、ＭＰＵＳＨ命令展開部２１は、オペランドなし（レジスタ指定がない）ＰＵＳＨ命令を生成し、そのコードをＲＡＭ１２に格納する（Ｓ３１）。次に、ＭＰＵＳＨ展開部２１は、ＭＰＵＳＨ命令のオペランドをチェックする（Ｓ３２）。ＭＰＵＳＨ命令展開部２１は、ＭＰＵＳＨ命令に記述されたオペランドを順にチェックし、未処理のオペランドがあれば（Ｓ３２，ＯＫ）、そのオペランドを含んだＰＵＳＨ命令を生成してそのコードをＲＡＭ１２に格納するとともに、そのオペランドをＬＩＦＯ２４に格納

する（Ｓ３３）。そして、ステップＳ３２へ戻って以降の処理を繰り返す。また、未処理のオペランドがなければ（Ｓ３２，ＮＧ）、ＰＵＴ命令を生成してそのコードをＲＡＭ１２に格納する（Ｓ３４）。

【００９７】

このようにして、ＭＰＵＳＨ命令が展開されて、そのコードがＲＡＭ１２に格納される。たとえば、図１３（ａ）の（１）に示すＭＰＵＳＨ命令の場合であれば、ステップＳ３３においてまずオペランドＲ０が抽出されて、“ｐｕｓｈ　Ｒ０”のコードが生成される。次に、オペランドＡＲ０が抽出されて、“ｐｕｓｈ　ＡＲ０”のコードが生成される。なお、図１６に示すＬＩＦＯ２４は、図１２に示すＲＡＭ１２またはハードディスク１３に形成される。

【００９８】

図１７は、図１５のステップＳ５（ＭＰＯＰ命令の展開）の処理をさらに詳細に説明するためのフローチャートである。まず、ＭＰＯＰ命令展開部２２は、オペランドなし（レジスタ指定がない）ＰＯＰ命令を生成し、そのコードをＲＡＭ１２に格納する（Ｓ５１）。次に、ＭＰＯＰ展開部２２は、ＭＰＵＳＨ命令の展開時にＬＩＦＯ２４に格納されたオペランドを読み出し（Ｓ５２）、未処理のオペランド（レジスタ）の有無を判定する（Ｓ５３）。

【００９９】

ＭＰＯＰ命令展開部２２は、未処理のオペランドがあれば（Ｓ５３，ＯＫ）、そのオペランドを含んだＰＯＰ命令を生成してそのコードをＲＡＭ１２に格納する（Ｓ５４）。そして、ステップＳ５２へ戻って以降の処理を繰り返す。また、未処理のオペランドがなければ（Ｓ５３，ＮＧ）、処理を終了する。なお、図１７に示すＬＩＦＯ２４は、図１２に示すＲＡＭ１２またはＣＰＵ１０内のレジスタ群に相当する。

【０１００】

このようにして、ＭＰＯＰ命令が展開されて、そのコードがＲＡＭ１２に格納される。たとえば、図１３（ａ）の（２）に示すＭＰＯＰ命令の場合であれば、ＬＩＦＯ２４に“Ｒ０”および“ＡＲ０”が格納されているので、ステップＳ５４においてまずオペランドＡＲ０が抽出されて、“ｐｏｐ　ＡＲ０”のコードが

生成される。次に、オペランドＲ０が抽出されて、“ｐｏｐ　Ｒ０”のコードが生成される。

【０１０１】

以上説明したように、本実施の形態におけるアセンブラによれば、マクロ命令を実施の形態１において説明したマイクロプロセッサが実行可能なコードに展開するようにしたので、退避および復帰すべきレジスタを含んだマクロ命令を記述するだけで一連のスタック操作を行なうコードを生成することが可能となる。したがって、プログラマはスタック操作の整合性等の確認を行なう必要がなくなり、ソフトウェア開発における生産性を向上させることが可能となった。

【０１０２】

（実施の形態３）

上述した実施の形態２のアセンブラにおいては、ワークレジスタＴＲ０がレジスタ転送の媒介として使用される。図１１の説明から分かるように、ワークレジスタＴＲ０の退避の有無に関わらず、ワークレジスタＴＲ０の値が自動的にスタックに格納される。本実施の形態におけるアセンブラは、この特徴を利用したものである。

【０１０３】

本実施の形態におけるアセンブラの機能的構成は、図１４に示す実施の形態２におけるアセンブラの機能的構成と比較してＭＰＵＳＨ命令展開部の機能のみが異なる。また、本実施の形態におけるアセンブラの処理手順は、図１５に示す実施の形態２におけるアセンブラの処理手順と同じである。したがって、重複する構成および機能の詳細な説明は繰り返さない。なお、本実施の形態におけるＭＰＵＳＨ命令展開部の参照符号を２１’として説明する。

【０１０４】

図１８は、本実施の形態におけるアセンブラによって処理されるマクロ命令を説明するための図である。図１８（ａ）の（１）に示すマクロ命令“ＭＰＵＳＨ　ＴＲ０，ＡＲ０；”は、ワークレジスタＴＲ０およびアドレスレジスタＡＲ０を退避することを示している。なお、“ＭＰＵＳＨ”以降に退避すべきレジスタが記述されるものとし、レジスタの数は特に限定がないものとする。

【０１０５】

図１８（ａ）の（２）に示すマクロ命令“ＭＰＯＰ”は、直前に記述されたマクロ命令“ＭＰＵＳＨ”に対応するマクロ命令であり、マクロ命令“ＭＰＵＳＨ”によって退避されたレジスタの内容を全て復帰させるものである。なお、マクロ命令の記述方法はこれらに限られるものでなく、マクロ命令が展開された後の命令コードが均等であれば同一と考えられるべきである。

【０１０６】

図１８（ｂ）は、図１８（ａ）に示すマクロ命令“ＭＰＵＳＨ”を展開したときの命令コードを示している。また、図１８（ｃ）は、図１８（ａ）に示すマクロ命令“ＭＰＯＰ”を展開したときの命令コードを示している。ワークレジスタＴＲ０はスタックに自動的に格納されることになるため、ワークレジスタＴＲ０に対応した退避命令は生成されない。また、ワークレジスタＴＲ０以外のレジスタを復帰させると、自動的にワークレジスタＴＲ０も復帰されるため、ワークレジスタＴＲ０に対応した復帰命令も生成されない。

【０１０７】

図１９は、図１５のステップＳ３（ＭＰＵＳＨ命令の展開）の処理をさらに詳細に説明するためのフローチャートである。まず、ＭＰＵＳＨ命令展開部２２’は、オペランドなし（レジスタ指定がない）ＰＵＳＨ命令を生成し、そのコードをＲＡＭ１２に格納する（Ｓ６１）。次に、ＭＰＵＳＨ展開部２１’は、ＭＰＵＳＨ命令のオペランドをチェックする（Ｓ６２）。

【０１０８】

ＭＰＵＳＨ命令展開部２１’は、ＭＰＵＳＨ命令に記述されたオペランドを順にチェックし、未処理のオペランドがあれば（Ｓ６２，ＯＫ）、そのオペランドがワークレジスタＴＲ０であるか否かを判定する（Ｓ６３）。そのオペランドがワークレジスタＴＲ０であれば（Ｓ６３，Ｙｅｓ）、そのままステップＳ６２へ戻って以降の処理を繰り返す。

【０１０９】

また、オペランドがワークレジスタＴＲ０でなければ（Ｓ６３，Ｎｏ）、そのオペランドを含んだＰＵＳＨ命令を生成してそのコードをＲＡＭ１２に格納する

とともに、そのオペランドをＬＩＦＯ２４に格納する（Ｓ６４）。そして、ステップＳ６２へ戻って以降の処理を繰り返す。また、未処理のオペランドがなければ（Ｓ６２，ＮＧ）、ＰＵＴ命令を生成してそのコードをＲＡＭ１２に格納し（Ｓ６５）、処理を終了する。

【０１１０】

以上説明したように、本実施の形態におけるアセンブラによれば、レジスタとメモリとの間のデータ転送において媒介として使用されるレジスタを退避／復帰する命令を生成しないようにしたので、レジスタの退避／復帰における冗長なコードが生成されるのを防止することができ、プログラムのステップ数の削減、処理速度の向上および使用メモリサイズの削減を図ることが可能となった。

【０１１１】

今回開示された実施の形態は、すべての点で例示であって制限的なものではないと考えられるべきである。本発明の範囲は上記した説明ではなくて特許請求の範囲によって示され、特許請求の範囲と均等の意味および範囲内でのすべての変更が含まれることが意図される。

【０１１２】

【発明の効果】

請求項１に記載のマイクロプロセッサによれば、１つの命令コードでレジスタ間のデータ転送と、レジスタとメモリとの間のデータ転送とを実行できるため、所定の処理を行なうプログラムのステップ数を削減することが可能となった。

【０１１３】

請求項２に記載のマイクロプロセッサによれば、直接メモリにデータを退避できない第２のレジスタに格納されたデータの退避を、１命令分のスループットで実行することが可能となった。

【０１１４】

請求項３に記載のマイクロプロセッサによれば、第２のレジスタに格納されたデータを第１のレジスタに転送した後、スタックポインタの値がデクリメントされるため、スタック操作をさらに容易に行なうことが可能となった。

【０１１５】

請求項４に記載のマイクロプロセッサによれば、直接メモリからデータを復帰できない第２のレジスタへのデータの復帰を、１命令分のスループットで実行することが可能となった。

【０１１６】

請求項５に記載のマイクロプロセッサによれば、メモリに格納されたデータを第１のレジスタに転送した後、スタックポインタの値がインクリメントされるため、スタック操作をさらに容易に行なうことが可能となった。

【０１１７】

請求項６に記載のマイクロプロセッサによれば、第２のレジスタとメモリとの間のデータ転送を、ワークレジスタを媒介として行なうことが可能となった。

【０１１８】

請求項７に記載のマイクロプロセッサによれば、第１のレジスタを媒介として制御レジスタとメモリとの間のデータ転送を行なうことが可能となった。また、第１のレジスタをワークレジスタとしてデータ演算手段に実装し、第２のレジスタを制御レジスタとしてアドレス演算手段または制御手段に実装することによって、ＲＩＳＣ方式のマイクロプロセッサの回路構成を採用することが可能となった。

【０１１９】

請求項８に記載のマイクロプロセッサによれば、第１のレジスタに格納されたデータをメモリに転送し、スタックポインタの値をそのまま保持する退避命令を実行可能とすることにより、スタックポインタが最後に退避されたデータを指し示すようにすることが可能となった。

【０１２０】

請求項９に記載のアセンブラによれば、第１のコード生成手段は、第１のマクロ命令から複数のレジスタを退避するコードを生成するので、１つのマクロ命令で複雑なスタック操作を扱うことが可能となった。また、第２のコード生成手段は、記憶手段に記憶された複数のレジスタを特定する情報を参照して、複数のレジスタを復帰させるコードを生成するので、１つのマクロ命令で複雑なスタック操作を扱うことができ、複数のレジスタの退避／復帰の整合性を自動的に管理す

ることが可能となった。

【０１２１】

請求項１０に記載のアセンブラによれば、レジスタとメモリとの間のデータ転送の媒介として使用されるレジスタが自動的にスタックに退避される場合に、冗長なコードが生成されるのを防止することが可能となった。

【０１２２】

請求項１１に記載のアセンブリ方法によれば、第１のマクロ命令から複数のレジスタを退避するコードを生成するので、１つのマクロ命令で複雑なスタック操作を扱うことが可能となった。また、読み込まれたコードが第２のマクロ命令である場合、記憶された複数のレジスタを特定する情報を参照して、複数のレジスタを復帰させるコードを生成するので、１つのマクロ命令で複雑なスタック操作を扱うことができ、複数のレジスタの退避／復帰の整合性を自動的に管理することが可能となった。

【０１２３】

請求項１２に記載のアセンブリ方法によれば、レジスタとメモリとの間のデータ転送の媒介として使用されるレジスタが自動的にスタックに退避される場合に、冗長なコードが生成されるのを防止することが可能となった。

【０１２４】

請求項１３に記載の記録媒体によれば、第１のマクロ命令から複数のレジスタを退避するコードを生成するので、１つのマクロ命令で複雑なスタック操作を扱うことが可能となった。また、読み込まれたコードが第２のマクロ命令である場合、記憶された複数のレジスタを特定する情報を参照して、複数のレジスタを復帰させるコードを生成するので、１つのマクロ命令で複雑なスタック操作を扱うことができ、複数のレジスタの退避／復帰の整合性を自動的に管理することが可能となった。

【０１２５】

請求項１４に記載の記録媒体によれば、レジスタとメモリとの間のデータ転送の媒介として使用されるレジスタが自動的にスタックに退避される場合に、冗長なコードが生成されるのを防止することが可能となった。

【図面の簡単な説明】

【図１】　本発明の実施の形態１におけるマイクロプロセッサを構成するレジスタセットを説明するための図である。

【図２】　本発明の実施の形態１におけるマイクロプロセッサの概略構成を示すブロック図である。

【図３】　本発明の実施の形態１におけるマイクロプロセッサのパイプライン処理を説明するための図である。

【図４】　本発明の実施の形態１におけるマイクロプロセッサによって処理される演算命令の一例を示す図である。

【図５】　本発明の実施の形態１におけるマイクロプロセッサによって処理される転送命令、シーケンス制御命令および特殊命令の一例を示す図である。

【図６】　ＬＯＡＤ命令およびＳＴＯＲＥ命令において指定可能なレジスタの一覧を示す図である。

【図７】　ＳＴＯＲＥ命令を用いたレジスタの退避動作を行なうプログラムの一例を示す図である。

【図８】　図７に示すプログラムが実行されたときのスタックの動作を示す図である。

【図９】　ＰＯＰ命令、ＰＵＳＨ命令およびＰＵＴ命令のニーモニックおよびその動作を説明するための図である。

【図１０】　ＰＯＰ命令、ＰＵＳＨ命令およびＰＵＴ命令を使用したプログラムの一例を示す図である。

【図１１】　図１０に示すプログラムを実行したときのスタックの動作を説明するための図である。

【図１２】　本発明の実施の形態２におけるアセンブラを実現するコンピュータの構成例を示すブロック図である。

【図１３】　本発明の実施の形態２におけるアセンブラによって処理されるマクロ命令を説明するための図である。

【図１４】　本発明の実施の形態２におけるアセンブラの概略構成を示すブロック図である。

【図１５】　本発明の実施の形態２におけるアセンブラの処理手順を説明するためのフローチャートである。

【図１６】　図１５のステップＳ３の処理をさらに詳細に説明するためのフローチャートである。

【図１７】　図１５のステップＳ５の処理をさらに詳細に説明するためのフローチャートである。

【図１８】　本発明の実施の形態３におけるアセンブラによって処理されるマクロ命令を説明するための図である。

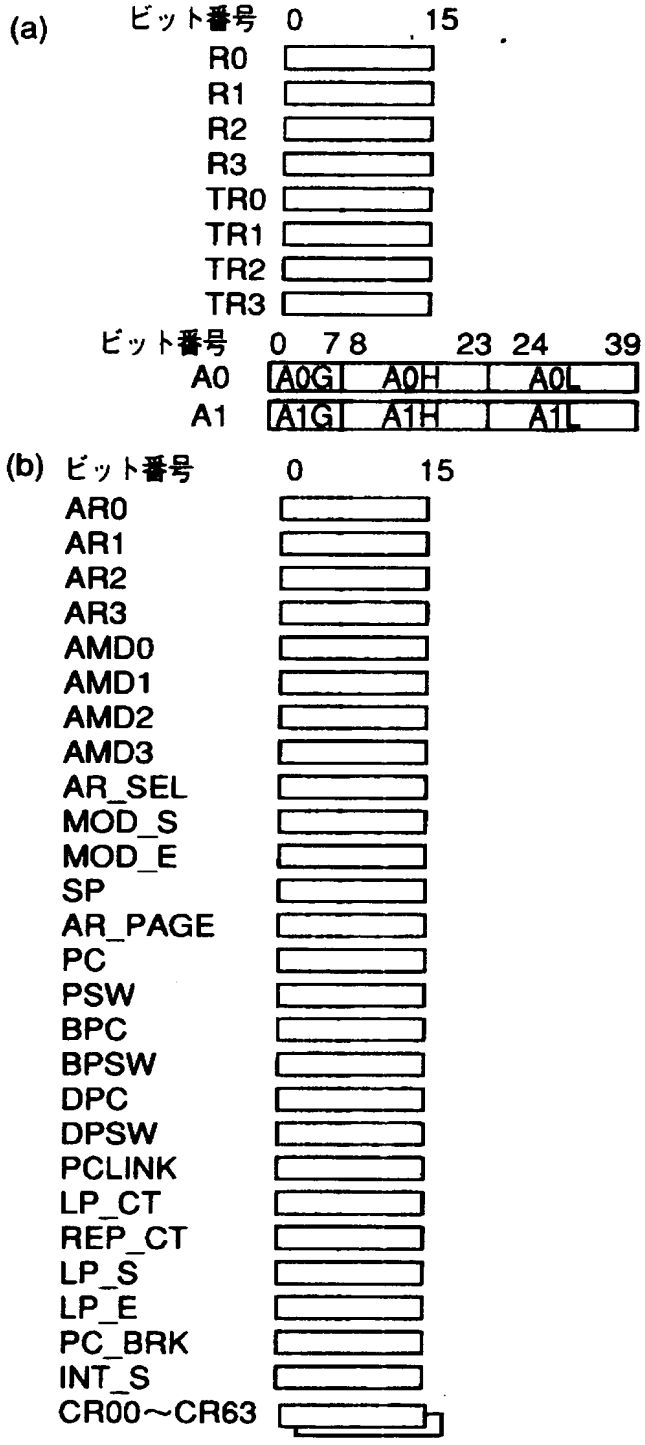
【図１９】　図１５のステップＳ３の処理をさらに詳細に説明するためのフローチャートである。

【符号の説明】

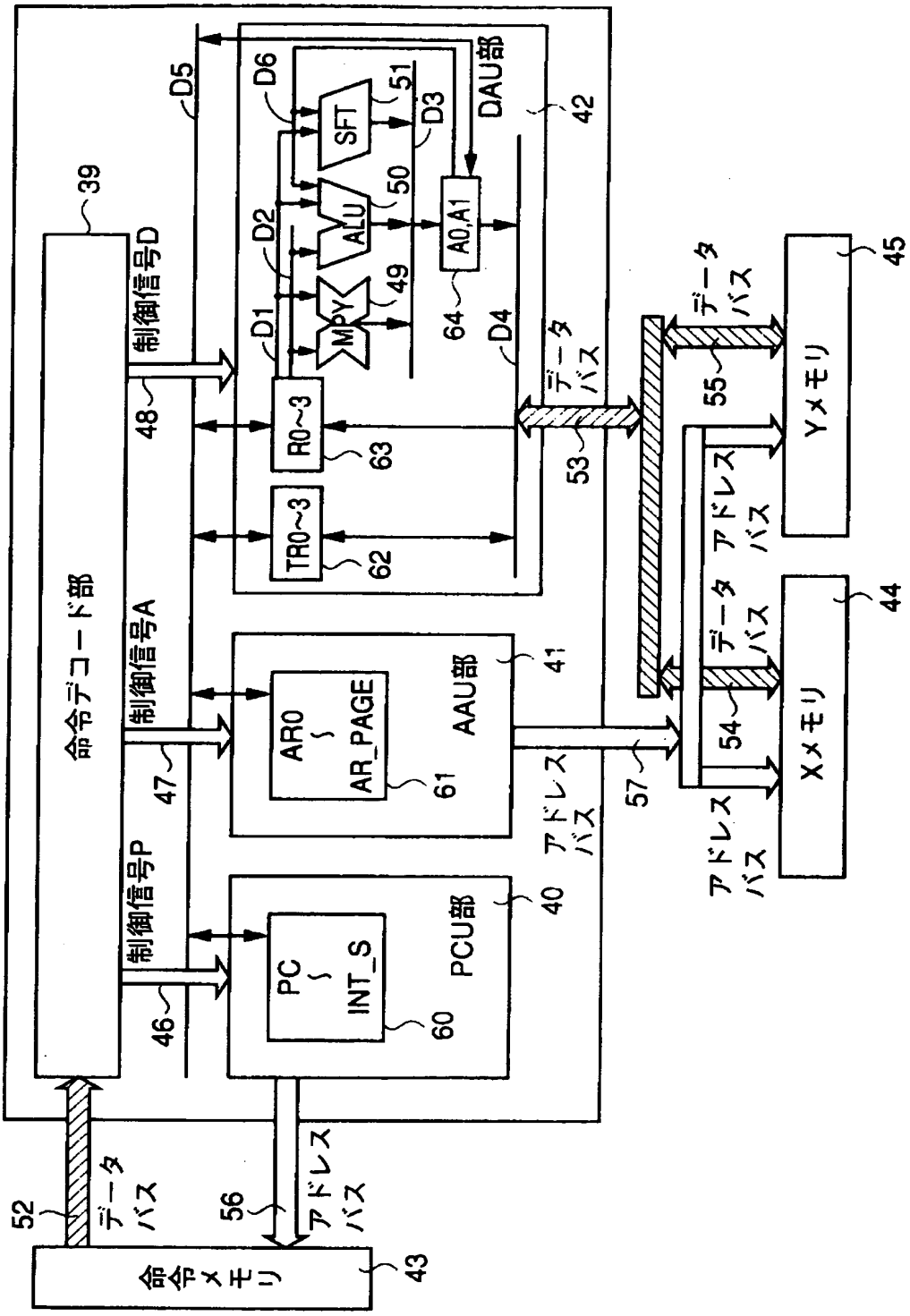
１　コンピュータ本体、２　グラフィックディスプレイ装置、３　ＦＤドライブ、４　ＦＤ、５　キーボード、６　マウス、７　ＣＤ－ＲＯＭ装置、８　ＣＤ－ＲＯＭ、９　ネットワーク通信装置、１０　ＣＰＵ、１１　ＲＯＭ、１２　ＲＡＭ、１３　ハードディスク、２０　コード解釈部、２１　ＭＰＵＳＨ命令展開部、２２　ＭＰＯＰ命令展開部、２３　コード生成部、２４　ＬＩＦＯ、３９　命令デコード部、４０　ＰＣＵ部、４１　ＡＡＵ部、４２　ＤＡＵ部、４３　命令メモリ、４４　Ｘメモリ、４５　Ｙメモリ、４９　乗算器、５０　ＡＬＵ、５１　シフタ。

【書類名】　図面

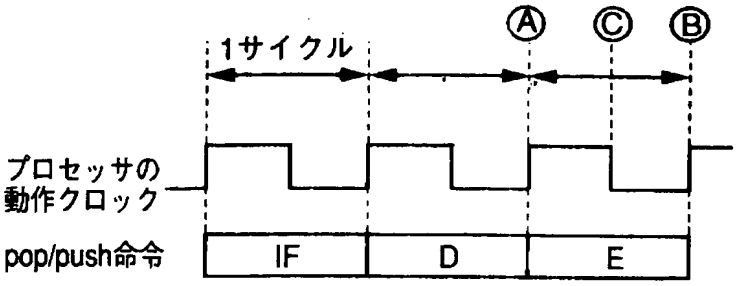
【図１】


(a)
ビット番号 0 15
R0
R1
R2
R3
TR0
TR1
TR2
TR3
ビット番号 0 7 8 23 24 39
A0
A0G A0H A0L
A1
A1G A1H A1L
(b) ビット番号 0 15
AR0
AR1
AR2
AR3
AMD0
AMD1
AMD2
AMD3
AR_SEL
MOD_S
MOD_E
SP
AR_PAGE
PC
PSW
BPC
BPSW
DPC
DPSW
PCLINK
LP_CT
REP_CT
LP_S
LP_E
PC_BRK
INT_S
CR00～CR63

【図２】


命令デコード部
39
制御信号D
制御信号A
制御信号P
48
47
46
DAU部
42
SFT
ALU
MPY
A0,A1
R0~3
TR0~3
D1
D2
D3
D4
D5
D6
49
50
51
62
63
64
53
データバス
アドレスバス
AAU部
41
AR0
AR_PAGE
61
PCU部
40
PC
INT_S
60
57
54
55
52
56
Yメモリ
45
Xメモリ
44
命令メモリ
43

【図３】



【図４】

演算命令	
mul	Multiply
muluu	Multiply unsigned operands
mac	Multiply and add
macuu	Multiply unsigned operands and add
macsu	Multiply signed operand by unsigned operand and add
macsul	Multiply signed operand by unsigned operand and add with shift right
macsuh	Multiply signed operand by unsigned operand and add with shift left
msub	Multiply and sub
msubuu	Multiply unsigned operands and sub
msubsul	Multiply signed operand by unsigned operand and add with shift right
msubsuh	Multiply signed operand by unsigned operand and add with shift left
add	Add a register to acc high
addl	Add a register to acc low
sub	Subtract a register from acc high
subl	Subtract a register from acc low
min	Set minimum value of acch or reg to accumulator
max	Set maximum value of accl or reg to accumulator
amin	Set minimum value to dest-acc
amax	Set maximum value to dest-acc
sra	Shift arithmetic right or left an accumulator
srl	Shift logical right or left an accumulator
and	And
or	Or
xor	Xor
nop	No operation
trfh	Transfer to an accumulator high
trfl	Transfer to an accumulato low
trf	Transfer to an accumulator
aadd	Add accumulators
asub	Subtract src-acc from dest-acc
sadd	Add dest-acc and src-acc with shift
abs	Absolute an accumulator
neg	Negate an accumulator
test	Test an accumulator(acc<0:set Nflag, acc==0:set Zflag)
md	Round an accumulator
not	Not an accumulator

【図５】

転送命令

mv	Copy one word from a register to a register
ldi	Load immediate
ld	Load
st	Store
push	Push to stack
put	Put to stack
pop	Pop from stack

シーケンス制御命令

jmp	Jump
call	Jump & link
loopi	Set loop counter and start hardware DO loop
loop	Start hardware DO loop
repeati	Set repeat counter and repeat next instruction
repeat	Repeat next instruction
return	Return from subroutine
reit	Return from EIT
rtd	Return from debugger EIT

特殊命令

adr_set	Set AR_SEL register
mvin	Move from IO registers
mvout	Move to IO registers
slave	Transit to slav mode
noop	No operation

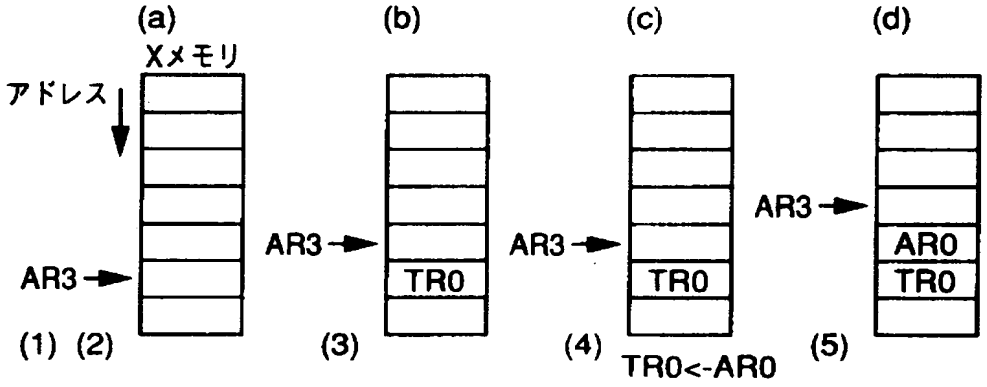
【図６】

	指定可能なレジスタ
LOAD命令	R0, R1, R2, R3, TR0, TR1, TR2, TR3
STORE命令	TR0, TR1, TR2, TR3, A0H, A0L, A1H, A1L

【図７】

```
LDI AR3, #STACK_BOTTOM    ; (1)
LDI AMD3, #DEC_1          ; (2)
  ・ ・ ・ ・ ・ ・ ・
ST TR0, X:AR3             ; (3)
MV TR0, AR0               ; (4)
ST TR0, X:AR3             ; (5)
```

【図８】



【図９】

(a) POP

[mnemonics]

(1) pop

(2) pop ra

[operation]

(1) tr0 = x_memory[sp];
sp++;

(2) ra = tr0;
tr0 = x_memory[sp];
sp++;

(b) PUSH

[mnemonics]

(1) push

(2) push ra

[operation]

(1)
sp--;

(2) x_memory[sp] = tr0;
tr0 = ra;
sp--;

(c) PUT

[mnemonics]

put

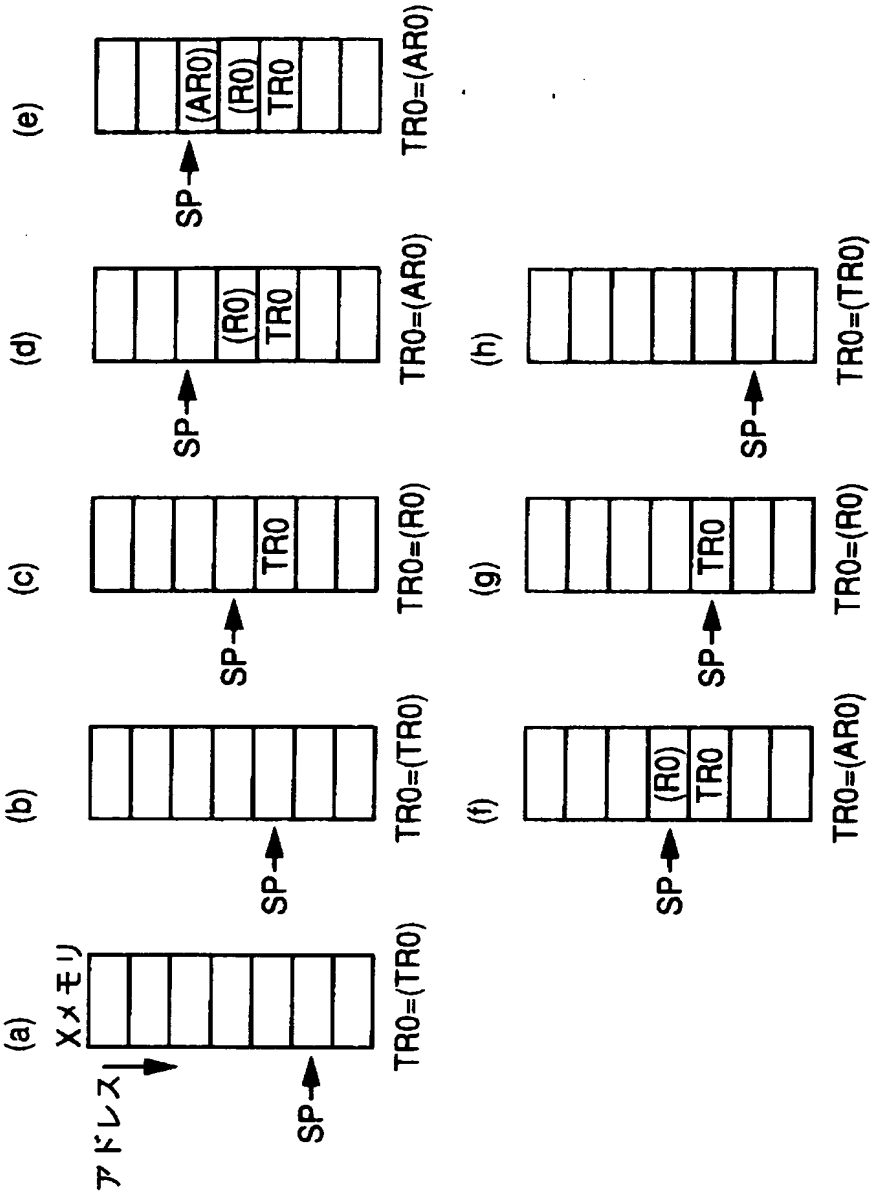
[operation]

x_memory[sp] = tr0;

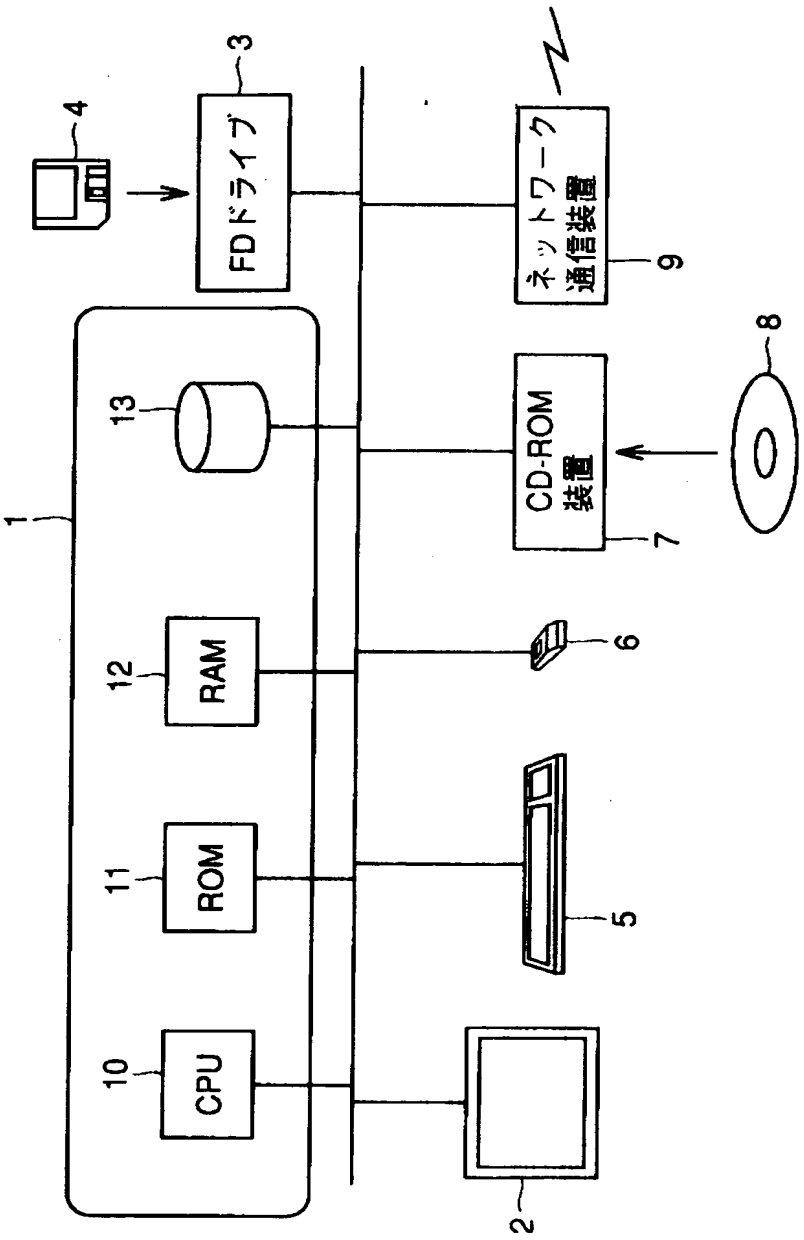
【図１０】

```
push            ; (1)
push  R0        ; (2)
push  AR0       ; (3)
put             ; (4)
 . . . . . . . . . .
pop             ; (5)
pop  AR0        ; (6)
pop  R0         ; (7)
```

【図１１】


(a)
メモリ
アドレス
SP
TR0=(TR0)
(b)
SP
TR0=(TR0)
(c)
TR0
SP
TR0=(R0)
(d)
(R0)
TR0
SP
TR0=(AR0)
(e)
(AR0)
(R0)
TR0
SP
TR0=(AR0)
(f)
(R0)
TR0
SP
TR0=(AR0)
(g)
TR0
SP
TR0=(R0)
(h)
SP
TR0=(TR0)

【図１２】

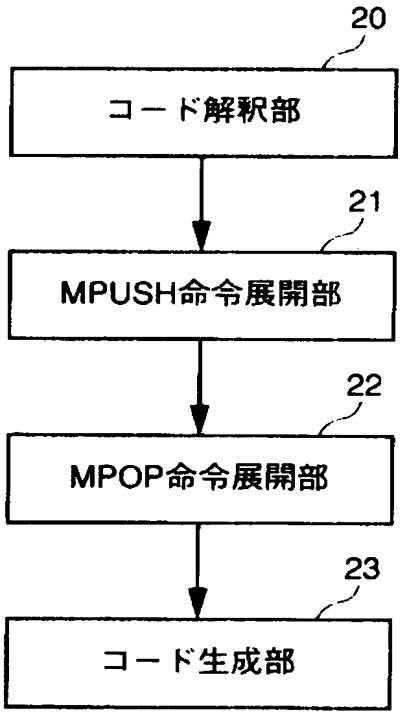


【図１３】

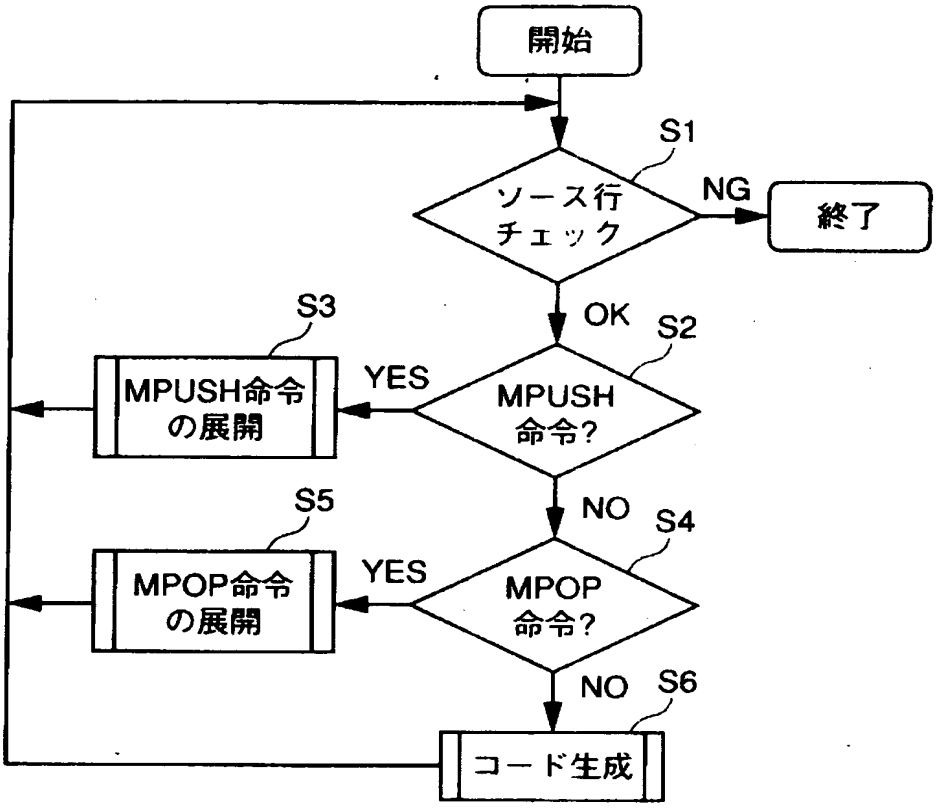
```
(a)
        MPUSH  R0, AR0; (1)
        . . . . . .
        MPOP   ;(2)
(b)
  push          ;
  push  R0      ;
  push  AR0     ;
  put           ;

(c)
  pop           ;
  pop  AR0      ;
  pop  R0       ;
```

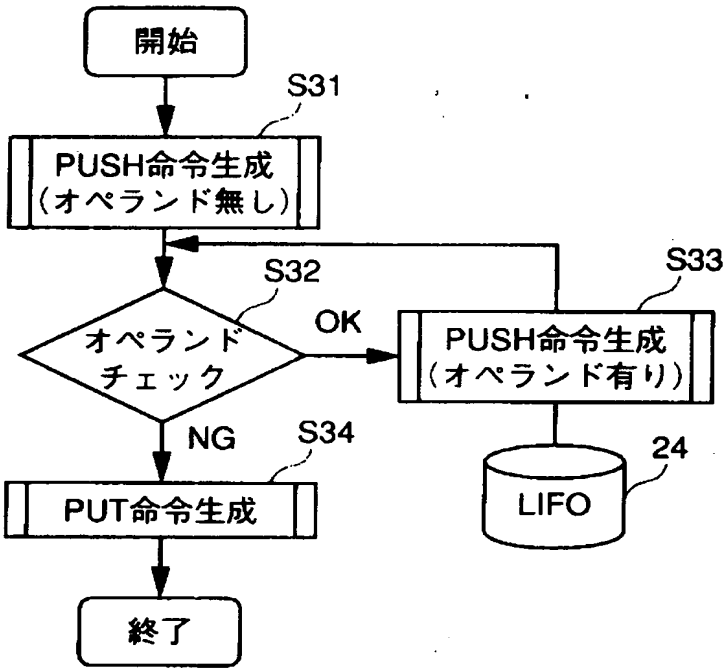
【図１４】



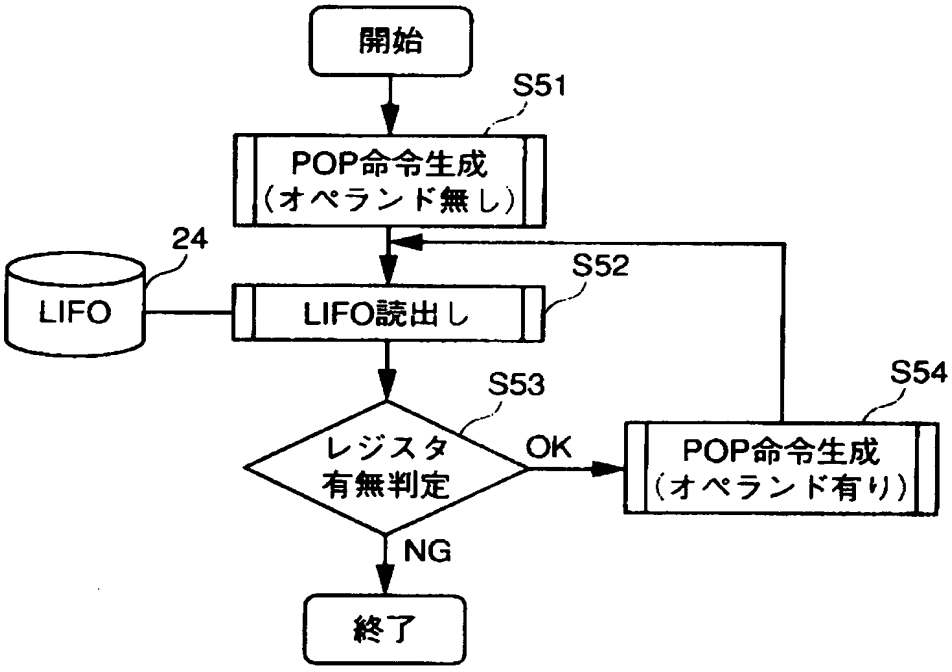
【図１５】


開始
S1
ソース行
チェック
NG
終了
OK
S2
MPUSH
命令?
YES
S3
MPUSH命令
の展開
NO
S4
MPOP
命令?
YES
S5
MPOP命令
の展開
NO
S6
コード生成

【図１６】


開始
S31
PUSH命令生成
(オペランド無し)
S32
S33
オペランド
チェック
OK
PUSH命令生成
(オペランド有り)
NG
S34
24
LIFO
PUT命令生成
終了

【図１７】

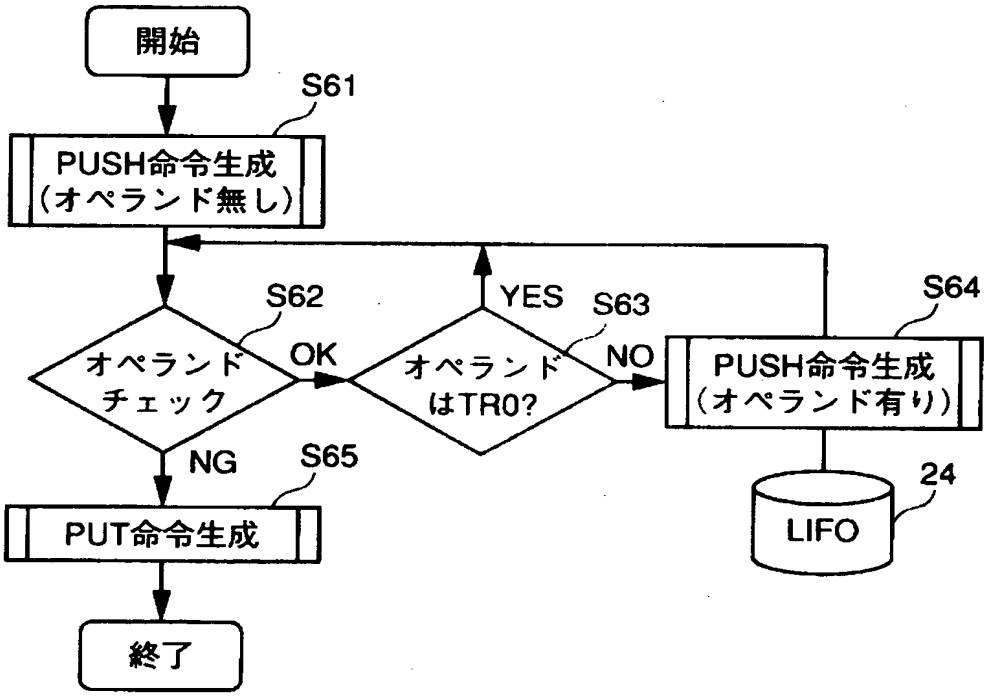

開始
S51
POP命令生成
(オペランド無し)
24
LIFO
S52
LIFO読出し
S53
S54
レジスタ
有無判定
OK
POP命令生成
(オペランド有り)
NG
終了

【図１８】

```
(a)
        MPUSH  TR0, AR0; (1)
        . . . . . .
        MPOP   ;(2)
(b)
  push          ;
  push  AR0     ;
  put           ;

(c)
  pop           ;
  pop   AR0     ;
```

【図１９】



【書類名】　　要約書

【要約】

【課題】　複数の制御レジスタの退避／復帰を短いステップ数のプログラムで実現できるマイクロプロセッサを提供すること。

【解決手段】　命令コードのフェッチを制御するＰＣＵ部４０と、フェッチされた命令コードをデコードする命令デコード部３９と、命令デコード部３９によるデコード結果に基づいて、メモリのアドレスを演算するＡＡＵ部４１と、１つの退避命令に対応して、制御レジスタとワークレジスタとの間のデータ転送と、ワークレジスタとＸメモリ４４との間のデータ転送とを行なうＤＡＵ部４２とを含む。したがって、直接メモリにデータを退避できない制御レジスタに格納されたデータの退避を、１つの退避命令で実行することが可能となる。

【選択図】　　図２

出願人履歴情報

識別番号　〔０００００６０１３〕

１．変更年月日　１９９０年　８月２４日
［変更理由］　新規登録
住　所　東京都千代田区丸の内２丁目２番３号
氏　名　三菱電機株式会社